A ROBUST AND SCALABLE SOFTWARE LIBRARY FOR PARALLEL ADAPTIVE REFINEMENT ON UNSTRUCTURED MESHES

John Z. Lou, Charles D. Norton, and Thomas A. Cwik
National Aeronautics and Space Administration
Jet Propulsion Laboratory, California Institute of Technology
MS 168-522, 4800 Oak Grove Drive, Pasadena, CA 91109-8099, U.S.A.
{John.Lou, Charles.Norton, Thomas.Cwik}@jpl.nasa.gov

Abstract

The design and implementation of **Pyramid** (http://www-hpc.jpl.nasa.gov/APPS/AMR/), a software library for performing parallel adaptive mesh refinement (PAMR) on unstructured meshes, is described. This software library can be easily used in a variety of unstructured parallel computational applications, including parallel finite element, parallel finite volume, and parallel visualization applications using triangular or tetrahedral meshes. The library contains a suite of well-designed and efficiently implemented modules that perform operations in a typical PAMR process. Among these are mesh quality control during successive parallel adaptive refinement (typically guided by a local-error estimator), parallel load-balancing, and parallel mesh partitioning using the ParMeTiS partitioner. The Pyramid library is implemented in Fortran 90 with an interface to the Message-Passing Interface (MPI) library, supporting code efficiency, modularity, and portability. An EM waveguide filter application, adaptively refined using the Pyramid library, is illustrated.

1. Introduction

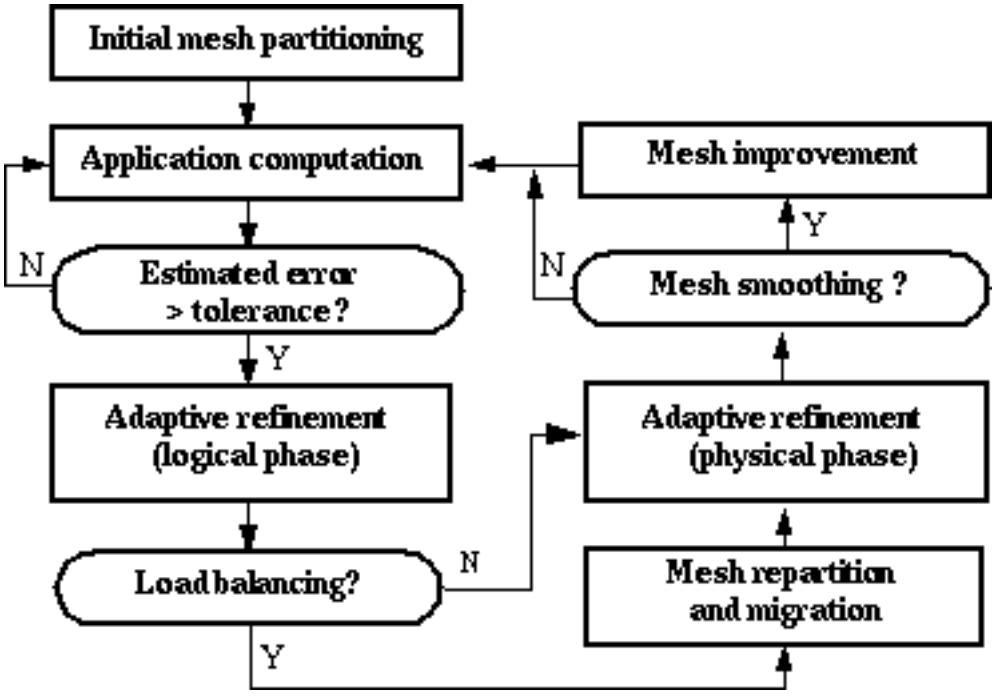
Adaptive mesh refinement (AMR) represents a class of numerical techniques that has demonstrated great effectiveness for a variety of computational applications including computational physics, structural mechanics, electromagnetics, and semiconductor device modeling. When an application domain is discretized into a computational mesh, various portions of the mesh can be refined, or coarsened, in regions where varying degrees of accuracy are required. This approach saves memory and computing time over methods that use a uniform resolution over the entire application domain.

Unfortunately, the development of an efficient and robust adaptive mesh refinement component for an application, particularly for unstructured meshes on multiprocessor systems, is very complex. The motivation for our work is to provide an efficient and robust parallel AMR library that can be easily integrated into unstructured parallel applications. Therefore, our library approach separates support for parallel adaptive refinement and mesh maintainence techniques from any application-specific solution processes.

Research on parallel AMR for unstructured meshes has been previously reported [1,8]. Most efforts are based on C++, and many realize that mesh quality control during successive adaptive refinement is an active research topic. Our work features the use of Fortran 90 and MPI for parallel AMR on unstructured triangular and tetrahedral meshes, the implementation of robust schemes for parallel adaptive refinement and mesh quality control during a repeated AMR process, and a "plug-in" component for stiffness matrix construction in finite element applications. We selected

Reprinted from NASA HPCCP Computational Aerosciences Workshop, pages 241-246, NASA Ames Research Center, Moffett Field, CA, January 1999.

Fortran 90 for our implementation because it provides new abstraction modeling facilities beneficial for parallel unstructured AMR development. This approach also simplifies interface concerns with scientific application codes, many of which were developed in Fortran 77 for high performance.



General organization of the parallel AMR process for unstructured meshes.

2. The AMR Components

The general organization of the parallel AMR process is illustrated. Initially, the (generally random) input mesh must be repartitioned and redistributed after loading from the disk. The application computation and local error-estimation step occur, after which a logical AMR process occurs. (Load balancing can occur based on this process since the refinement scheme is completely defined, although it has not yet physically occurred.)

The load balancing process moves coarse elements from the logical refinement, based on a weighting scheme, to the proper destination processors using the migration module. At this point, the physical AMR step occurs by applying local refinement processes.

Finally, the element quality can be checked by performing an explicit mesh smoothing operation or by ensuring high quality element creation during refinement. We apply the latter approach since it prevents degradation of mesh quality after successive adaptive refinements. Every stage of our AMR process is performed using parallelism.

3. Fortran 90 and Abstraction Modeling Principles in Parallel AMR Development

Fortran 90 modernizes traditional Fortran 77 scientific programming by adding many new features. These features allow programs to be designed and written at a higher level of abstraction, while increasing software clarity and safety without sacrificing performance [7]. Fortran 90's capabilities encourage scientists to design new kinds of advanced data structures supporting complex applications, like parallel AMR. These capabilities extend beyond the well-known array syntax and dynamic memory management operations.

While Fortran 90 is not an object-oriented language (certain OO features can be emulated by software constructs) the methodology simplifies library interfaces such that the internal details are hidden from library users [5]. Fortran 90 modules and derived types allow user-defined types, like the mesh, to be defined with associated routines. Modules that capture essential features of parallel AMR can be combined with each other, adding to the flexibility and organization of the software design. These techniques, and other features, allow the library to contain clearly organized interfaces for use in parallel applications.

4. The Adaptive Refinement Process

The adaptive refinement process is based on logical and physical refinements. The logical refinement step uses an iterative procedure that traverses through elements of the coarse mesh repeatedly to "define" a consistent mesh refinement pattern on the coarse mesh. The result of the logical refinement is stored in the data structure of the coarse mesh, which completely specifies whether and how each element in the coarse mesh should be refined. Our adaptive refinement scheme is based on "edge-marking" for both triangular and tetrahedral meshes. Starting from a predetermined subset of elements, the logical refinement scheme proceeds by marking (or logically refining) element edges wherever necessary, and the refinement pattern for each element is determined by the number of marked edges in that element.

With information generated from the logical refinement step, the actual mesh refinement becomes conceptually simpler, since it is completely specified how each element should be refined. To make the physical refinement process simpler and efficient, low-level objects are refined before refining high-level objects. On a triangular mesh, it means edges are refined before refining elements.

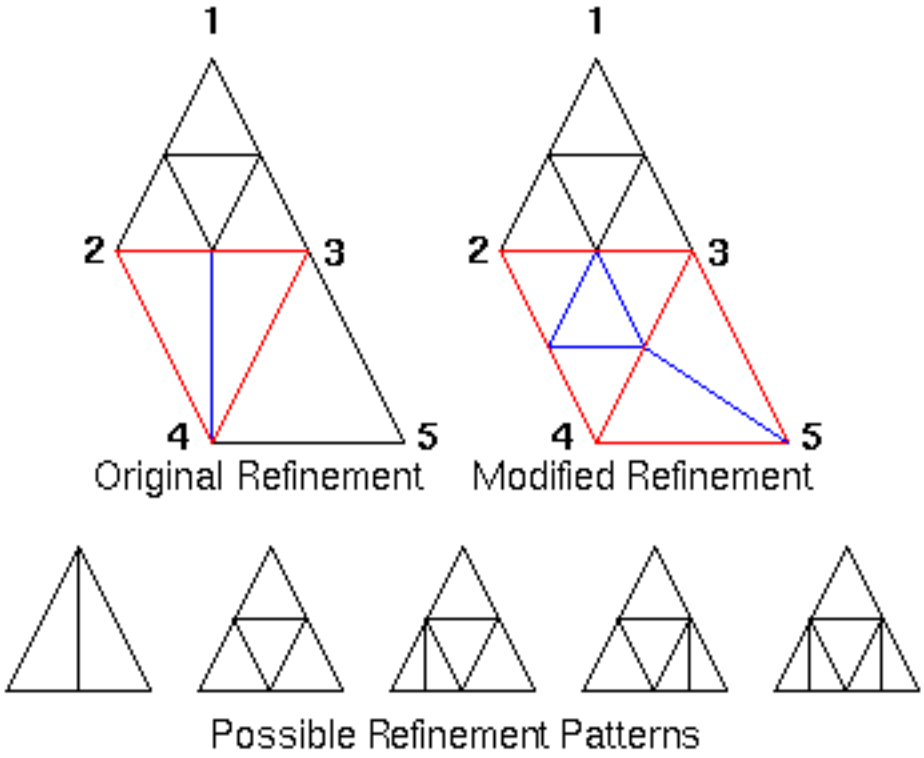
To perform a parallel logical adaptive refinement, we extend the serial scheme so that after traversing the local element set for edge-marking, each processor updates the status of edges on mesh partition boundaries by exchanging the edge status information (i.e. marked or not marked) with its neighboring processors.

5. Mesh Quality Control

A problem associated with repeated AMR operations, typically guided by a local-error estimator, is the deterioration of mesh quality. Most mesh smoothing schemes tend to change the structure of a given mesh to achieve the "smoothing effect" by rearranging nodes in the mesh. The changes made by a smoothing scheme, however, could modify the desired distribution of element density produced by the AMR procedure, and the cost of performing a global mesh smoothing could be very high. Nevertheless, applying a relatively efficient smoothing scheme over the last adaptively refined mesh is probably reasonable for mesh quality improvement. Alternatively, it is possible to

prevent, or slow down, the degradation of element quality during a repeated adaptive refinement process.

The mesh quality control scheme we have applied classifies elements based on how they were refined. This allows us to forsee the potential of creating elements with poor aspect ratios in the next refinement. After identifying those elements, we can replace them with a refinement pattern that improves upon the geometry.



The figure shows the original refinement of a coarse element (2-3-4). Successive refinements will destroy the aspect ratio of existing elements, leading to poor mesh quality. The approach we apply modifies the coarse element refinement, as shown, should either of the child elements require further refinement (due to local errors or mesh consistency constraints from neighbor element refinement). This process controls the mesh quality, at the slight expense of creating more elements.

We integrate the mesh quality control feature into our adaptive refinement scheme, for triangular meshes, by defining all possible refinement patterns for a pair of “twin” transitional elements (child elements of element 2-3-4 in the original mesh). During the logical refinement step the scheme checks all marked edges of the twin elements allowing one of the indicated refinement patterns to be selected. To simplify the physical refinement stage we ensure that the partitioner will not place the twin elements onto different processors. This guarantees that once mesh migration has occured, the physical refinement for the parent element (2-3-4) will create child elements in a local manner. Refinement patterns are also applied for tetrahedral meshes as well.

6. Interlanguage Communication and Load Balancing Issues

Our software needs to communicate with the ParMeTiS parallel mesh partitioner, which is written in the C programming languauge [6]. We have a single routine that acts as the conduit between our Fortran 90 system and the C ParMeTiS library. Interlanguage communication between Fortran 90 and C is not a problem, provided the linker knows the format of external routine names (generally, underscore_, doubleunderscore__, UPPERCASE, or lowercase). Fortran 90 derived type

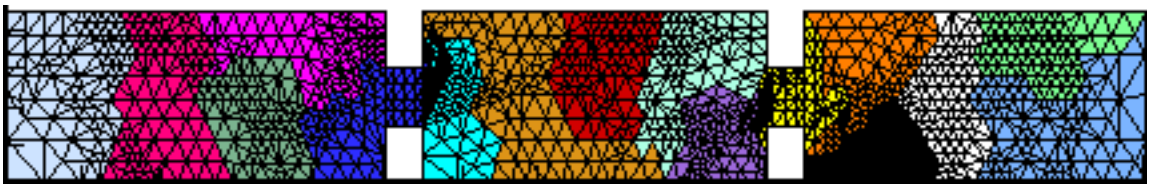
objects can be passed by reference to C structures, or simple arrays can be communicated, but we advise using the Fortran 90 SEQUENCE attribute to request the proper byte-alignment ordering. The weighted graph helps ParMeTiS attempt to minimize element movement and the number of components on partition boundaries.

This process involves converting the distributed mesh into a distributed graph by computing the dual of the mesh. When the partitioner returns, a mapping for every element is specified. The migration module redistributes the elements among the processors based on this mapping. Since the communication is irregular, and unpredictable, an efficient non-blocking irregular communication scheme has been developed for the element redistribution. In the final stage, the mesh data structure is reconstructed using efficient heap-sorting techniques. This entire process occurs in a parallel and distributed manner.

7. Application to an EM Waveguide Filter

Our AMR library has been tested in the finite-element simulation of electromagnetic wave scattering in a waveguide filter [4]. The problem is to solve Maxwell's equation for the electromagnetic (EM) fields in the filter domain. A local-error estimate procedure based on the Element Residue Method (ERM) is used in combination with the AMR technique to adaptively construct an optimal mesh for the problem solution.

The adaptive refinement and partitioning of a finite element mesh for EM scattering in the waveguide filter is illustrated on 16 processors of the NASA Goddard Cray T3E. An example of adaptive refinement for a 3D tetrahedral mesh is available.



Adaptive refinement, mesh partitioning, and migration applied to a waveguide filter.

8. Summary

A complete framework for performing parallel adaptive mesh refinement in unstructured applications on multiprocessor computers has been described. This includes a robust parallel AMR scheme, mesh quality control, load-balancing, the implementation technique using Fortran 90 and MPI, and the interlanguage communication issues. Electromagnetic scattering in a waveguide filter has been demonstrated. Parallel performance results on several multiprocessor systems will be given in our final paper. More information on Pyramid: A JPL Parallel Unstructured AMR Library is also available.

The Table 1 gives performance results of the AMR (logical and physical) step and the load balancing and migration step. The refinement randomly chooses half of the elements per processor. The number of elements increases with the partitioning slightly due to maintinain mesh consistency constrains based on this refinement scheme.

Acknowledgments

The research described in this paper was performed at Jet Propulsion Laboratory, California Institute of Technology, under contract to the National Aeronautics and Space Administration. The supercomputers used in this work were provided with funding from the NASA offices of Space Science, Aeronautics, and Mission to Planet Earth.

Table 1: Results for Waveguide Filter after 3 Refinements on the NASA Goddard Cray T3E

# of Processors	AMR Time	Load Balancing (Migration) Time	Number of Elements
32	57.34 sec	15.36 sec	292,612
64	13.55 sec	3.75 sec	295,405
128	2.93 sec	1.65 sec	305,221
256	0.54 sec	1.51 sec	335,527
512	0.27 sec	1.86 sec	397,145

References

1. R. Biswas, L. Oliker, and A. Sohn. “Global Load-Balancing with Parallel Mesh Adaption on Distributed-Memory Systems.” Proceedings of Supercomputing `96, Pittsburgh, PA, Nov. 1996.

2. E. Boender. “Reliable Delaunay-Based Mesh Generation and Mesh Improvement.” Communications in Numerical Methods in Engineering, Vol. 10, 773-783 (1994).

3. Graham F. Carey, “Computational Grid Generation, Adaptation, and Solution Strategies”. Series in Computational and Physical Processes in Mechanics and Thermal Science. Taylor & Francis, 1997.

4. T. Cwik, J. Z. Lou, and D. S. Katz, “Scalable Finite Element Analysis of Electromagnetic Scattering and Radiation.” to appear in Advances in Engineering Software, V. 29 (2), March, 1998.

5. V. K. Decyk, C. D. Norton, and B. K. Szymanski. “Expressing Object-Oriented Concepts in Fortran 90”. ACM Fortran Forum, vol. 16, num 1, pp. 13-18, April 1997. Also as NASA Tech Briefs, Vol. 22, No. 3, pp 100-101, March 1998 (reduced version).

6. G. Karypis, K. Schloegel, and V. Kumar. “ParMeTiS: Parallel Graph Partitioning and Sparse Matrix Ordering Library Version 1.0”. Tech. Rep., Dept. of Comp. Science, U. Minnesota, 1997.

7. C. Norton, V. Decyk, and B. Szymanski. “High Performance Object-Oriented Scientific Programming in Fortran 90”. Proc. 8th SIAM Conf. on Parallel Proc. for Sci. Comp., Mar. 1997.

8. M. Shephard, J. Flaherty, C. Bottasso, H. de Cougny, C. Ozturan, and M. Simone. “Parallel automatic adaptive analysis”. Parallel Computing 23 (1997) pg. 1327-1347.